CSCI 3353 Object Oriented Design
Homework Assignment 5
Due Monday, March 12

I have written an animated fish tank for you to enjoy, which is in the file *fishtank.zip*. The zip file contains the necessary java files, as well as a folder containing image files of fish. In the folder are two images for several types of fish – one facing left, and one facing right.

The first thing you should do is get the code to work on your computer. The image folder needs to be placed, as is, in the appropriate spot in your file system. (On my computer, it is within the *src* directory of my Eclipse project.) The code uses the JavaFX library, which should be part of your Java SDK installation. If you are missing it, let me know ASAP. To add a new fish to the tank, click on the "New Fish" menu; to start and stop the animation, click on the "Animation" menu. Note that you can resize the fish tank and the fish will adapt.

The program is a lot of fun, but the fish are kind of boring. They all swim the same way, and at the same speed. In this assignment, you will use strategies and factories to customize how the fish are created. The assignment has several parts. It is a good idea to debug each part before beginning the next one.

NOTE: The code I am giving you is in the package *fishtank*. The code you write should be in the package *hw5*. It is good to have both packages. I found it very useful to be able to run the original code while I debugged my new code.

1. Add one (or more) additional fish types to the program. The *images* folder has many types to choose from. There are two steps to adding a new fish: adding an item to the menu bar; and modifying the code to create new *Fish* objects of that type. The method *Aquarium.createMenuBar* shows how fish types get added to the menu bar. Each menu item has a method *setOnAction*, whose argument is a lambda expression that says what to do when the button is clicked. Each lambda expression calls the *FishTank.addFish* method. You will need to modify *addFish* to handle the new type(s) of fish.

2. Currently, each type of fish is a subclass of the abstract class *Fish*. Refactor the code so that *Fish* is no longer abstract. Instead, create an interface *FishType*, and modify each of the subclasses so that they implement *FishType*. Each *Fish* object

should use a *FishType* object to get the fish's images; that *FishType* object is passed to the fish via its constructor.

3. Use the factory pattern to create a hierarchy of factory objects, headed by the abstract class *FishFactory*. There should be one factory object for each fish type. Each factory class should have a method *create* (with no arguments) that creates an instance of its associated *Fish* class. Then modify *Aquarium* to construct an array of these objects, and pass the array to *FishTank*. Modify the *FishTank.newFish* method so that it uses this array; the result should be that the method should no longer need to have any if-statements.

4. Currently, all fish behave the same. This behavior is determined by the values of the following variables in *Fish*: *xspeed*, *yspeed*, *xchangepct, ychangepct*, *xinitialpos*, and *yinitialpos*. Use the strategy pattern to extract these values into a strategy hierarchy, named *FishBehavior*. Each strategy class will assign its own values to these six variables. I would like you to write the following strategy classes:

- The class *NormalBehavior*, which consists of the values originally given.
- The class *ErraticBehavior*, in which a fish starts from a random location in the tank, moves up/down and left/right more frequently, and with greater speed.
- The class *BottomFeedingBehavior*, in which a fish moves slowly across the bottom of the tank.

Modify the *Fish* constructor so that it takes a *FishBehavior* object. Modify each fish factory to create a *FishBehavior* object. In particular, angel fish should have normal behavior, mean fish should have erratic behavior, and octopuses should have bottom-feeding behavior. Feel free to assign any behavior to the other fish type(s) you added to the program – either one of the above behaviors, or new ones that you create.

NOTE: I found it easier to make *FishBehavior* be an abstract class instead of an interface. That is completely acceptable.

5. Draw a class diagram depicting all of the classes in your new program. Save the diagram as pdf, in the file *HW5ClassDiagram.pdf*.

When you are done, create a zip file containing the pdf document and all of the Java files in your program, and submit it to Canvas.